Anonymeter: Unified Framework for Quantifying Privacy Risk in Synthetic Data

Table of Contents

How Anonymeter Works

The following sections provide a detailed explanation of how Anonymeter, the main framework of the paper under study, operates. If you want to see the part about my experiments, skip to - Experiments for Assignment 2 of Data Protection Technologies.

`Anonymeter` in a nutshel

In `Anonymeter` each privacy risk is derived from a privacy attacker whose task is to use the synthetic dataset to come up with a set of *guesses* of the form:

- "there is only one person with attributes X, Y, and Z" (singling out)
- "records A and B belong to the same person" (linkability)
- "a person with attributes X and Y also have Z" (inference)

Each evaluation consists of running three different attacks:

- the "main" privacy attack, in which the attacker uses the synthetic data to guess information on records in the original data.
- the "control" privacy attack, in which the attacker uses the synthetic data to guess information on records in the control dataset.
- the "baseline" attack, which models a naive attacker who ignores the synthetic data and guess randomly.

Checking how many of these guesses are correct, the success rates of the different attacks are measured and used to derive an estimate of the privacy risk. In particular, the "control attack" is used to separate what the attacker learns from the *utility* of the synthetic data, and what is instead indication of privacy leaks. The "baseline attack" instead functions as a sanity check. The "main attack" attack should outperform random guessing in order for the results to be trusted.

For more details, a throughout description of the framework and the attack algorithms can be found in the paper A Unified Framework for Quantifying Privacy Risk in Synthetic Data, accepted at the 23rd Privacy Enhancing Technologies Symposium (PETS 2023).

Setup and installation

`Anonymeter` requires Python 3.8.x, 3.9.x or 3.10.x installed. The simplest way to install `Anonymeter` is from `PyPi`. Simply run

```
pip install anonymeter
```

and you are good to go.

Local installation

To install `Anonymeter` locally, clone the repository:

```
git clone git@github.com:statice/anonymeter.git
```

and install the dependencies:

```
cd anonymeter  # if you are not there already
pip install . # Basic dependencies
pip install ".[notebooks]" # Dependencies to run example notebooks
pip install -e ".[notebooks,dev]" # Development setup
```

If you experience issues with the installation, we recommend to install `anonymeter` in a new clean virtual environment.

Getting started

Check out the example notebook in the `notebooks` folder to start playing around with `anonymeter`. To run this notebook you would need `jupyter` and some plotting libraries. This should be installed as part of the `notebooks` dependencies. If you haven't done so, please install them by executing:

```
pip install anonymeter[notebooks]
```

if you are installing anonymeter from `PyPi`, or:

```
pip install ".[notebooks]"
```

if you have opted for a local installation.

Basic usage pattern

For each of the three privacy risks anonymeter provide an `Evaluator` class. The high-level classes `SinglingOutEvaluator`, `LinkabilityEvaluator`, and `InferenceEvaluator` are the only thing that you need to import from `Anonymeter`.

Despite the different nature of the privacy risks they evaluate, these classes have the same interface and are used in the same way. To instantiate the evaluator you have to provide three dataframes: the original dataset `ori` which has been used to generate the synthetic data, the synthetic data `syn`, and a `control` dataset containing original records which have not been used to generate the synthetic data.

Another parameter common to all evaluators is the number of target records to attack (`n_attacks`). A higher number will reduce the statistical uncertainties on the results, at the expense of a longer computation time.

```
evaluator = *Evaluator(ori: pd.DataFrame,
                       syn: pd.DataFrame,
                       control: pd.DataFrame,
                       n_attacks: int)
```

Once instantiated the evaluation pipeline is executed when calling the `evaluate`, and the resulting estimate of the risk can be accessed using the `risk()` method.

```
evaluator.evaluate()
risk = evaluator.risk()
```

Configuring logging

`Anonymeter` uses the standard Python logger named `anonymeter`. You can configure the logging level and the output destination using the standard Python logging API (see here for more details).

For example, to set the logging level to `DEBUG` you can use the following snippet:

```
import logging

# set the logging level to DEBUG
logging.getLogger("anonymeter").setLevel(logging.DEBUG)
```

And if you want to log to a file, you can use the following snippet:

```
import logging

# create a file handler
file_handler = logging.FileHandler("anonymeter.log")

# set the logging level for the file handler
file_handler.setLevel(logging.DEBUG)

# add the file handler to the logger
logger = logging.getLogger("anonymeter")
logger.addHandler(file_handler)
logger.setLevel(logging.DEBUG)
```

Experiments for Assignment 2 of Data Protection Technologies

From this point forward, this readme will focus on the experiments conducted for Assignment 2 of Data Protection Technologies.

Prerequisites

Ensure you have Python installed. If not, download and install it from the official website.

Installing Dependencies

To install the required packages and libraries for the experiments, navigate to the repository's root directory and run the following command:

```
pip install -r requirements.txt
```

Code Contributions

All the code contributions made by me can be found in the `./src/custom_utility` directory. Navigate there to follow the rest of this README by running the following command in your terminal:

```
cd ./src/custom_utility
```

Reproducing Experiments

All the experiments conducted for this project can be found in the `./src/custom_utility/Utility_vs_Privacy.ipynb` Jupyter Notebook. To reproduce the results, follow these steps:

1. Open the Jupyter Notebook by navigating to the `./src/custom_utility` directory.
2. Launch Jupyter Notebook using the following command:

```
jupyter notebook Utility_vs_Privacy.ipynb
```

3. Within the notebook, you'll find detailed explanations and code for each experiment, including data preprocessing, K-anonymization, noise addition, and the evaluation of utility and privacy metrics.

Note: Running the Jupyter Notebook is sufficient to reproduce the experiments and view the results. You don't need to run individual modules or scripts separately for reproducing the experiments.

> As a further note, while this README provides an overview of the project, I will also explain other modules and scripts in detail to make everything clearer. This additional information will help you understand the project's structure and the functions of various components.

Feel free to explore the code, modify it as needed, and reproduce the experiments to gain a deeper understanding of the utility-privacy trade-offs in data protection technologies.

K-Anonymization

If you wish to perform K-anonymization on a dataset, you can do so using the `k_anonymize.py` script located in the `./src/custom_utility` directory. To anonymize a dataset, you must provide the following flags:

- `--K` : This flag is compulsory and specifies the value of K for K-anonymization.
- `--filename` : (Optional) Use this flag to specify the name of the dataset file you want to anonymize. By default, it assumes the data is located in the `root/data` directory.
- `--pathname` : (Optional) Use this flag to specify the path to your dataset if it's located outside the default directory.

Here's an example of how to run the script for K-anonymization:

```
python ./src/custom_utility/k_anonymize.py --K 5
--filename adults_syn_ctgan.csv --pathname ../../data/
```

Deanonymization:

The `de_anonymize.py` module restores original data with privacy protection intact. Here's how it works:

- Anonymeter framework and RandomForestClassifier require anonymized data to retain original values. `de_anonymize.py` replaces generalized values with the most frequent ones from the original data, making it suitable for models and privacy evaluation while still being anonymized.

- Configure the module in `deanonymization_config.json` to tailor the deanonymization process to your needs.

Adding Noise

The `add_noise.py` module introduces controlled noise to your data, enhancing privacy while maintaining data utility. Here's how it works:

- You provide a DataFrame to `add_noise.py`, specifying which columns should receive noise and at what level (low, medium, or high).
- For numerical attributes, Gaussian noise with configurable standard deviations (1, 5, or 10) is added. Categorical data is probabilistically modified with chances of 0.05, 0.1, or 0.3.
- The result is a privacy-enhanced dataset with a degree of noise, perfect for protecting sensitive information.

Computing Utility

The `compute_utility.py` module plays a crucial role in assessing the performance of your altered (anonymized or noisy) datasets. Here's how it works:

- You provide an altered dataset to `compute_utility.py`, which can be either anonymized or noisy.
- The module uses a model trained on the original data to evaluate the dataset's performance. Metrics such as accuracy (acc), F1 score (f1), and Receiver Operating Characteristic Area Under Curve (ROC AUC) are computed.
- By comparing these metrics with the results from the original test data, you gain insights into the utility of the transformations. This helps you understand the trade-off between privacy enhancement and data utility.

Thank You!

Happy experimenting!